

27-Dynamic Method Dispatch

Runtime Polymorphism in Java

Definition:

Runtime polymorphism, also known as Dynamic Method Dispatch, is a process where a call to an overridden method is resolved at runtime rather than compile-time. This mechanism allows Java to determine which method implementation to invoke based on the actual object type, rather than the reference type.

How It Works:

In dynamic method dispatch, an overridden method is called through the reference variable of a superclass. The method that gets executed is determined by the object that the reference variable points to at runtime.

Simplified Example

Here's a simpler way to achieve the same output i.e. same runtime object and type of object reference:

```
1 class A {
2     public void show() {
3         System.out.println("In A's Show");
4     }
5 }
6
7 class B extends A {
8     public void show() {
9         System.out.println("In B's Show");
10    }
11 }
12
13 public class Demo {
14     public static void main(String[] args) {
15         A obj = new A(); // Reference and object of type A
16         obj.show(); // Outputs: In A's Show
17     }
18 }
19
```

Output:

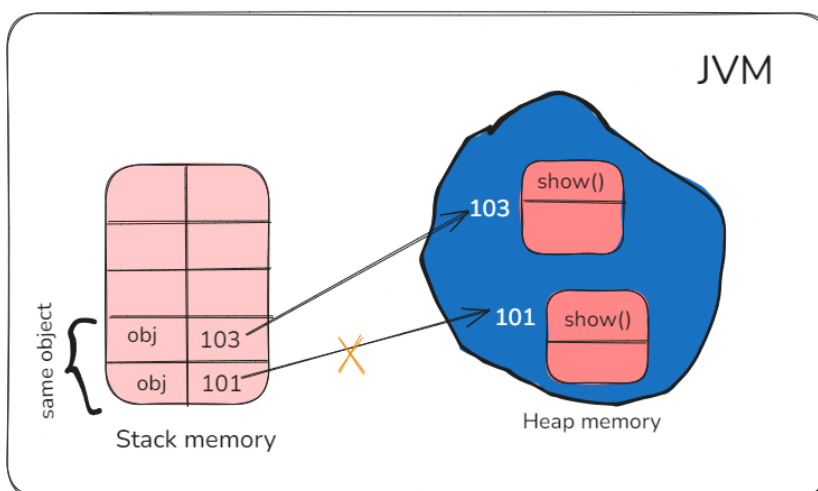
Output	Generated Files
In A's Show	

Here's an example to demonstrate dynamic method dispatch:

```
1 class A {  
2     public void show() {  
3         System.out.println("In A's Show");  
4     }  
5 }  
6  
7 class B extends A {  
8     public void show() {  
9         System.out.println("In B's Show");  
10    }  
11 }  
12  
13 public class Demo {  
14     public static void main(String[] args) {  
15         A obj = new A(); // Reference and object of type A  
16         obj.show(); // Outputs: In A's Show  
17  
18         obj = new B(); // Reference of type A, but object of type B  
19         obj.show(); // Outputs: In B's Show  
20     }  
21 }
```

Output:

Output	Generated Files
In A's Show In B's Show	



Explanation:

- Initially, obj is a reference to an object of type A. The show() method from class A is invoked.
- Later, same obj is assigned a new object of type B. Now, the show() method of class B is invoked, demonstrating dynamic method dispatch.

Example that demonstrates how the same reference of type A can invoke methods from classes B, and C based on the object it points to at runtime:

```
1 class A {
2     public void show() {
3         System.out.println("In A's Show");
4     }
5 }
6
7 class B extends A {
8     public void show() {
9         System.out.println("In B's Show");
10    }
11 }
12
13 class C extends A {
14     public void show() {
15         System.out.println("In C's Show");
16     }
17 }
18
19 public class Demo {
20     public static void main(String[] args) {
21         // Reference of type A pointing to an object of type A
22         A obj = new A();
23         obj.show(); // Outputs: In A's Show
24
25         // Reference of type A pointing to an object of type B
26         obj = new B();
27         obj.show(); // Outputs: In B's Show
28
29         // Reference of type A pointing to an object of type C
30         obj = new C();
31         obj.show(); // Outputs: In C's Show
32     }
33 }
```

Explanation:

1. Reference of Type A Pointing to an Object of Type A:

```
A obj = new A();
```

```
obj.show(); // Output: In A's Show
```

- Here, obj is both a reference of type A and an object of type A. The show() method from class A is invoked, so the output is "In A's Show".

2. Reference of Type A Pointing to an Object of Type B:

```
obj = new B();  
obj.show(); // Output: In B's Show
```

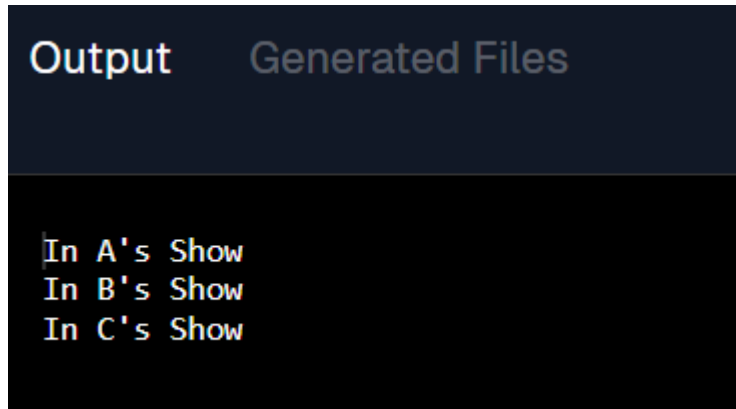
- The same reference obj is now assigned an object of type B. The show() method from class B is invoked, demonstrating dynamic method dispatch. The output is "In B's Show".

3. Reference of Type A Pointing to an Object of Type C:

```
obj = new C();  
obj.show(); // Output: In C's Show
```

- Finally, the reference obj is assigned an object of type C. The show() method from class C is invoked, and the output is "In C's Show".

Output:



```
Output    Generated Files  
  
In A's Show  
In B's Show  
In C's Show
```

Key Points:

- **Dynamic Method Dispatch:** The method that gets executed is determined by the actual object type at runtime, even though the reference type remains the same (A in this case).
- **Polymorphism:** This example illustrates the concept of polymorphism in Java, where a single reference type can point to objects of different classes, and the method calls are resolved based on the actual object type.

Advantages of Dynamic Method Dispatch:

- **Supports Method Overriding:** Allows Java to support overriding of methods, which is essential for runtime polymorphism.
- **Flexibility:** A class can define common methods that will be used by all its subclasses, while subclasses can provide specific implementations of those methods.

- **Enhanced Functionality:** Subclasses can add specific methods, enhancing the functionality of the superclass.